Bilkent University
CS 319 Object-Oriented Software Engineering
Detailed Design Report
Group “Veni Vidi Code”

Group Members:

- Kerem ŞAHİN - 21901724
- Miray AYERDEM - 21901987
- Ferhat KORKMAZ - 21901940
- Melih Fazıl KESKİN - 21901831
- Kaan TEK - 21901946

1. Introduction

The software that is being developed by us allows various Bilkent members, such as students, instructors, and alumni, to access the sports centers around the campus more easily. By using this software, Bilkent members can make reservations from the East, Main, and Dormitories Sports Centers, register for the ongoing tournament, and register for courses that are available. Bilkent members can also request personal gym programs from sports center staff by using this software. To make these available, sports center staff can also maintain their ongoing workflow on that application. Sports center staff can check the other Bilkent members' attendance status, and if necessary they can restrict those that are violating Bilkent University's Gym Regulations. Also, the staff will be able to manage the reservations, registrations to tournaments, gym program requests, and available courses from the software. Moreover, another user type of the software is the Admin. They are responsible for managing account addition and deletion processes in Bilkent Sports Centers.

1.1 Purpose of the System

The aim of the System is to create a web-based application that allows Bilkent Sports Center staff to have a reduced workload by filtering and sorting among reservations, activities, and Bilkent members. Also, another aim is that the other users spend less time making reservations for sports activities when it is compared to the ongoing system. Furthermore, most of the users are less likely to need phone calls for reservations.

1.2 Design Goals

The application is mainly focusing on two design goals: Usability and Maintainability. Both of those goals are chosen since these criteria are significant not only for users and staff but also for the Bilkent Computer Center (BCC) personnel.

1.2.1 Usability

Since the aim of creating this system is to ease the management of the reservations and tournaments for Bilkent Sports Center staff and easing the appointments for the activities for

Bilkent members, a usable and practical system is required. Users of the system should benefit from the system more than the ongoing system. The system will consist of non-complex processes and UI components to enhance the experience of users. Also, whenever the user is required to enter input, they will be guided accordingly to the input type; such as if the required input is a type of a number, the user will not be able to enter a text. Whatever the screen size is, the font size will never be less than 14 pt, so that the texts will be easily readable by the user. In addition, the dimensions of the buttons never will be less than 7% of the screen. Moreover, the system will be simple and understandable as much as possible by implementing the UI according to the conventional methods that are common for many applications, and familiar to many users. React is a commonly used library for implementing websites and Material UI is providing website components that are familiar to most daily users in terms of visuals. Using components that are widely used will increase the usability overall since the users are more likely to have used them before. Also, the components that are gathered by Material UI have better functionalities in terms of handling the sizes and shapes of components on different devices and resolutions, which also increases usability. In order to find data, users will be able to search certain data (such as for reservations) that will enable them to find the specific information they are looking for. This will ensure that users will access their desired information easier and faster, which will increase the usability overall.

1.2.2 Maintainability

If this software ever deploys in real life, we, as a group, will not always be there to maintain it. Therefore, maintainable software design comes into place. To make that goal find its ultimate target, we are developing that application with modern software developing technologies as well as contemporary methods to develop the application. As Bilkent students, we had been analyzing the problem domain for years. As a result of that observation, we created an analysis for requirements by using universally validated UML diagrams such as Use-Case, class, sequence, and state diagrams. They give a clear idea about what actors are using that software, what relationship takes place between classes and objects, and how main operations occur during the use of the software. The answers to those questions are significantly enlightening for those that are going to maintain the software. Also, in the development, 3-Layer Architectural Style is used in order to achieve maintainability. Moreover, an actively used GitHub repository will guide

the BCC personnel as well. A famous Java framework, Spring Boot, MySQL, and React.js are used during the development by using well-commented and purifying spaghetti code for the backend and frontend purposes. All of these are well-documented technologies, which will be a guide as well.

2. High-level Software Architecture

2.1 Subsystem Decomposition

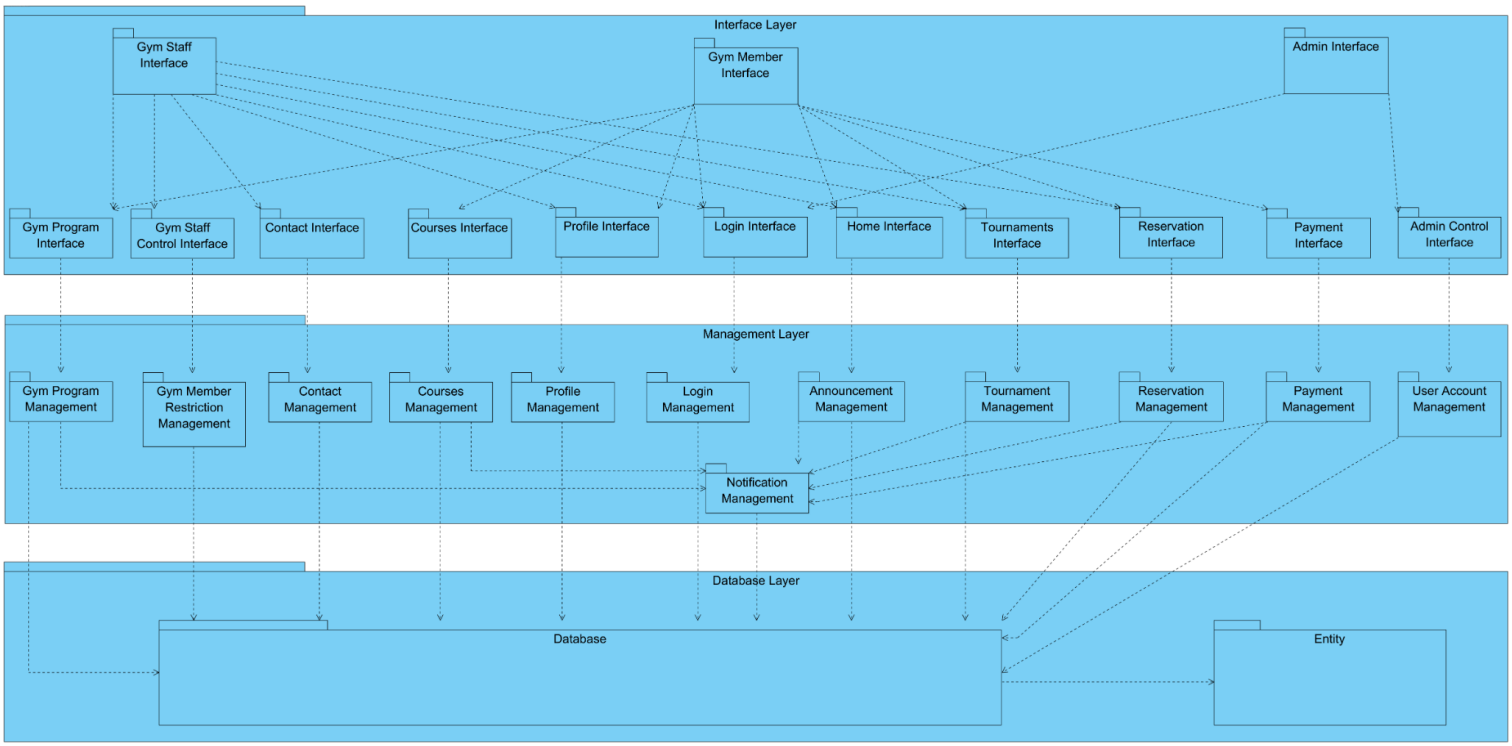

Fig. 1: Subsystem Decomposition Diagram of the System (Click here for bigger size)

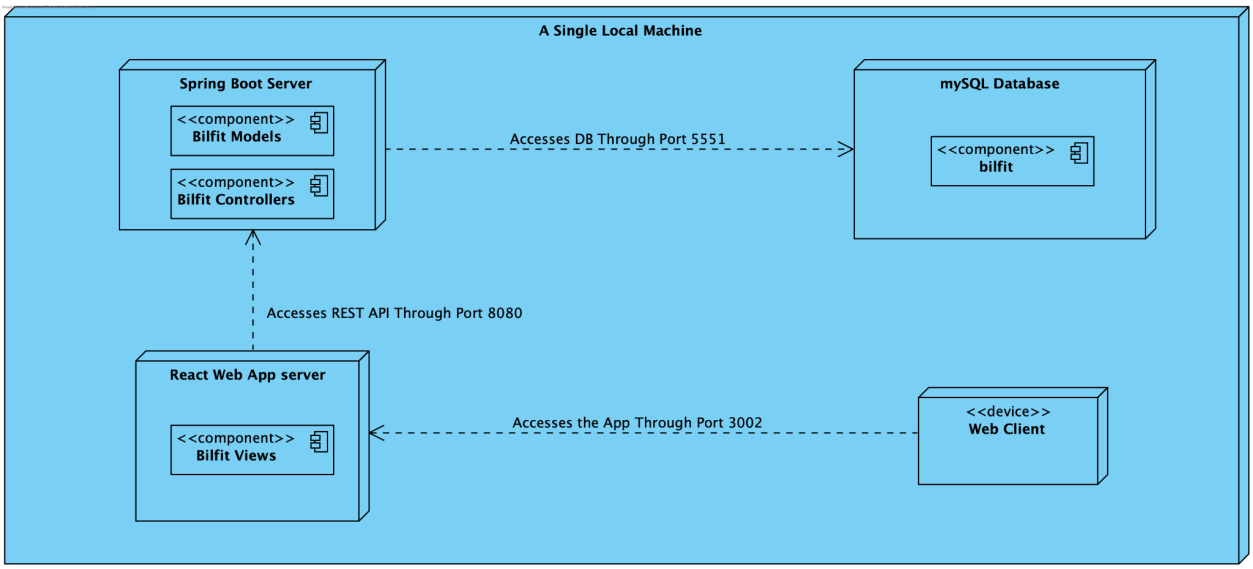

Fig. 2: Deployment Diagram

A 3-Layer Architecture style is chosen for the system. The 3-Layered Architecture style is a widely used architecture choice for the creation of interactive websites with multiple pages. It provides maintainability because it creates a hierarchy where the system is decomposed in such a way that it allows the developers to change and develop certain parts of the system without interfering with other parts of the system.

The first layer of the system which is called the “Interface Layer” is composed of the UI elements. The UI elements have been divided into three since different types of users will see slightly different web pages that are adapted according to the functionalities that they have. These UI elements will be responsible for displaying the content and also, will be responsible for getting input from users. Explanation of the interfaces of the Interface Layer:

- Gym Staff Interface: This interface is responsible for displaying the necessary interfaces that are going to be shown to the gym staff.
- Gym Member Interface: This interface is responsible for displaying the necessary interfaces that are going to be shown to the gym members.
- Admin Interface: This interface is responsible for displaying the necessary interfaces that are going to be shown to the Admin.

- Gym Program Interface: This interface will be responsible for displaying the necessary items in order to request gym programs (for GymMembers), and respond to the requests that are sent by the users (for GymStaff).
- Gym Staff Control Interface: This interface will be responsible for displaying the editorial parts of the different interfaces. For example, it will be responsible for displaying all of the reservations and also displaying items that allow the staff to edit, and cancel these reservations.
- Contact Interface: This interface is responsible for displaying the contact information about gym staff for gym members.
- Courses Interface: This interface is responsible for displaying the available courses for gym members and gym staff. Also, gym staff can manage the courses and gym members can enroll or cancel their courses on this interface.
- Login Interface: This interface is responsible for getting the necessary information from all types of users to sign in to the system.
- Home Interface: It will show the interface that will be the first interface that are going to be seen by users. It will welcome the users.
- Tournament Interface: This interface is responsible for displaying all available tournaments. Also, it provides gym staff to manage tournaments, and it provides gym members to enroll and cancel tournaments.
- Reservation Interface: This interface is responsible for displaying the necessary information to make or cancel reservations. Also, gym staff can see and manage users' reservations.
- Payment Interface: This interface is responsible for payment methodologies of Alumni and Instructors' Gym Member payment strategies.
- Admin Control Interface: This interface is responsible for displaying the information such as all users or adding new user features for admin.

The "Management Layer" of the system is responsible for getting the user's inputs and creating a connection between the UI elements and the database. It will be the main component that manipulates and operates on data. Also, it will receive information from the database, and it will transfer it to the UI elements for it to be displayed. Explanation of the interfaces of the Management Layer:

- Gym Program Management: It is the subsystem where the gym programs are created, edited, deleted, and shown.
- Gym Member Restriction Management: It is the subsystem where the Gym Members' allowance to the sports centers is controlled.
- Contact Management: It is the subsystem where the contact of the sports centers and their additional communication information is controlled.
- Courses Management: It is the subsystem where the courses are opened, edited, deleted, and shown.
- Profile Management: It is the subsystem where the Gym Members have access to edit their weight, height, phone number, and email.
- Login Management: It is the subsystem where the users (Gym Member, Gym Staff, And) login processes are handled. Their tokens are also created here.
- Announcement Management: It is the subsystem where the announcements are made by the GymStaff.
- Notification Management: It is the subsystem where the notifications to the specific user groups are sent following some actions such as reservations and tournament cancellation.
- Tournament Management: It is the subsystem where the tournaments are announced, edited, and deleted.
- Reservation Management: It is the subsystem where the Gym Members and Gym Staffs have bidirectional access to the reservations.
- User Account Management: It is the subsystem where Admins create and delete accounts of the Users. Also, this subsystem takes the role of changing the password requests of the users.
- Payment Management: It is the subsystem where payment information is taken, and the payment process is completed for the necessary activities.

The last "Database Layer" is composed of two subsystems, Database, and Entity. The database subsystem will be responsible for handling the database connection. It will be able to achieve this by using the repository interfaces which were inherited from Spring Boot

framework. The entity subsystem will be holding and managing the entity classes and their instances.

2.2 Hardware/Software Mapping

The system is going to be a lightweight website which does not require any specific hardware. Smartphones, tablets, and computers that are capable of running websites will be able to run the system. The basic needs for a computer that accesses the system include a monitor, keyboard, mouse, and a browser must be installed on the computer. Commonly used browsers such as Google Chrome (version: 14268.82), Mozilla Firefox (version: 99.0.1), Microsoft Edge (version: 100.0.1185.39).

The main purpose of React is to be fast, scalable, and simple. It works only on user interfaces in the application, which makes React a good choice for front-end development [1]. The choice for back-end software technologies is similar. As the developers are experienced in Java, Spring Boot framework is a reliable and easy-to-understand technology that also has rich resources such as MySQL support that we are using. The framework allows the usage of most architectural layers used in the development and allows simple setup and management, which is an important factor for our choice [2].

The hardware that is going to run MySQL, Spring Boot, and React Web App servers have the following properties:

- Apple M1 Pro processor with 8C CPU and 14C GPU
- 16 GB of Unified Memory
- macOS Monterey 12.3 Operating System
- 512 GB SSD

  OR
- Intel Core i7 -10750H processor
- 16 GB DDR4 RAM
- Windows 11 Pro
- 1 TB SSD

2.3 Persistent Data Management

In our application, on the backend side, we are using Spring Boot framework with MySQL Database. In order to make these two technologies more efficient, we are using Java Persistence API. Java Persistence API allows us to manage the MySQL database without writing complex SQL queries instead we are using Java annotations such as @Service, @Autowired, @Entity, @Id, @GeneratedValue, @GetMapping, @PostMapping, @PatchMapping, @PathVariable, and @RequestBody. These Java annotations directly fit into our Object-Oriented Java code and have more strong control over our database which stores user, reservation, sports center, tournament, course and other Entity objects, which can be seen more in detail in **section 3.2.3.2** [3].

2.4 Access Control and Security

The application stores crucial personal data like user ID, user email, user telephone, etc. so it is important to keep the data secure. For this reason, our application has different authentication levels for different application users. When users are created by the admin, each will have a user type which specifies their boundaries in the application. For example, Gym Staff will be able to see every gym member, but a gym member won't be able to have such functionality. Gym members also cannot see other gym members' and gym staff's personal information. Moreover, a gym staff cannot see other gym staffs' personal information as well. Only users created by the admin will be able to use the application, meaning that external users won't be able to access the website other than the login page, which increases the security. In order to keep the passwords secure, they will be hashed, and that is the way it will be stored in the database. Also, we will follow HTTPS Protocol in order to ensure security.

2.4.1 Access Matrix

	Admin	GymStaff	GymMember	Alumni	Student	Instructor
Login	x	x	x	x	x	x
Register User	x					
Delete User	x					
Edit User	x	x (self)	x (self)	x (self)	x (self)	x (self)
Restrict Gym Member		x				
Change Password		x	x	x	x	x
Make Reservation			x	x	x	x
Cancel Reservation		x	x	x	x	x
Pay For Reservation				x		x
Make Announcement		x				
Cancel Tournament		x				
Enroll To Tournament			x	x	x	x
Respond To Tournament Request			x	x	x	x
Cancel Tournament Registration			x	x	x	x
Open Course		x				
Edit Course		x				
Cancel Course		x				
Enroll To Course			x	x	x	x
Withdraw Course			x	x	x	x
Pay for Course Enrollment				x		x
Make Announcement		x				
See Notification		x	x	x	x	x
Send Gym Program		x				
Request Gym Program		x	x	x	x	x

Fig. 3: Access Matrix

2.5 Boundary Conditions

2.5.1 Initialization

For initialization, XAMPP program will be run to start the MySQL database. Then, in the source file, the com.venividicode.bilfit.BilfitApplication.java file will be run to make the backend available, which allows the REST API to work. The front-end web application's React project will be run with the command "npm start". Initialization approximately takes 2-3 minutes.

2.5.2 Termination

In order to successfully terminate the application, the reverse order of initialization should be followed to avoid any failure caused by termination. First, stop React Application by using the combination "control + c" in the terminal. Then, stop the Bilfit Application.java file. Lastly, stop the MySQL database by using the "stop" button while the database is highlighted.

2.5.3 Failure

For an unexpected event which causes an error in the system, we decided to direct the user to the login screen and terminate his login token. In this way, we will prevent any corruption in the database of the system by non-valid data or another problem. We believe this method will increase the security and reliability of the system. Of course, our aim is to have this kind of unexpected event to occur rarely and not to harm the user experience and usability. For most of the failure scenarios, it is possible to recover the app directly into the stable state since we are using three-layered architecture, meaning that it is impossible to go down unless the problem is in at least two layers.

3. Low-level Design

3.1 Object Design Trade-offs

Usability vs Functionality: One of the main goals of the system is to be easily usable by everyone. The method currently used by Bilkent Sports Center is time-consuming and therefore it is crucial that the system is easily usable. However, if there are a wide set of functionalities that are adopted by the system, it would be confusing for the user because they would have to remember all the subsections and functionalities of these different subsections. Therefore, in order to achieve usability, the functionalities of the program have to be restricted so that the users are going to be able to achieve what they want from the system without overcomplicating the capabilities of those functionalities.

Maintainability vs Efficiency: We used the 3-Layered architecture and various design patterns such as MVC, singleton, and strategy in order to make our application more maintainable. However, implementing our application in such a way led us to spend more time in order to write our code to follow those design patterns. Also, we had to put more effort to learn more about the design patterns and 3-Layered architecture, since they are not as easy and straightforward to implement compared to a code without any patterns or architectures being used. Therefore, trying to create a more maintainable system will decrease our efficiency in terms of time and effort spent.

Usability vs Security: Since the system is aimed to be usable, adding too much authentication would decrease the usability. However, these additional security steps are actually helpful in order to create a safer platform. There are security precautions that are taken in order to protect the data but these precautions are not extensive to the point that it decreases usability. So, considering the usability goal will restrict the extensive use of certain protection methods.

3.2 Final Object Design

3.2.1 Interface Layer

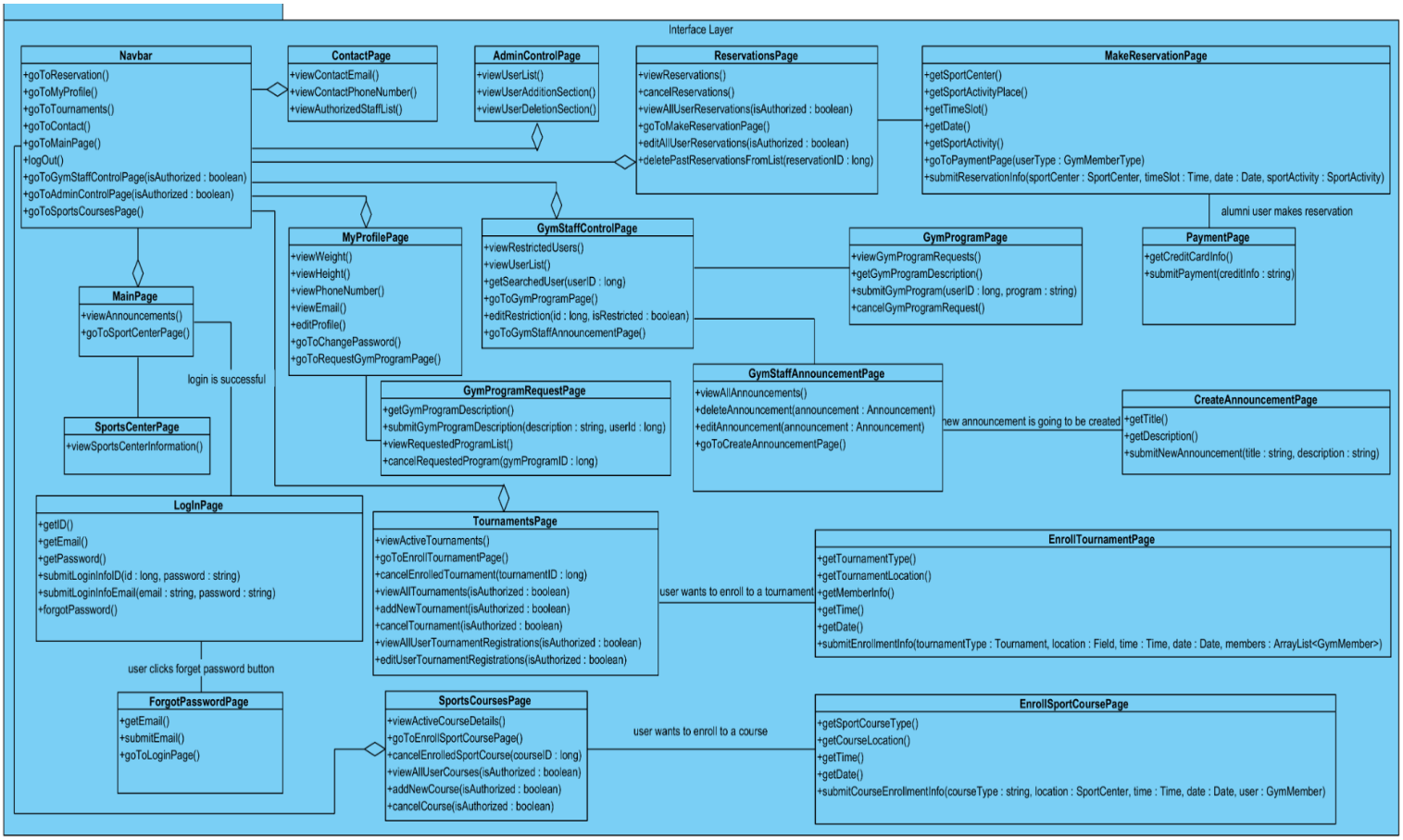

Fig. 4: Diagram of Classes in Interface Layer (Click here for bigger size)

3.2.2 Management Layer

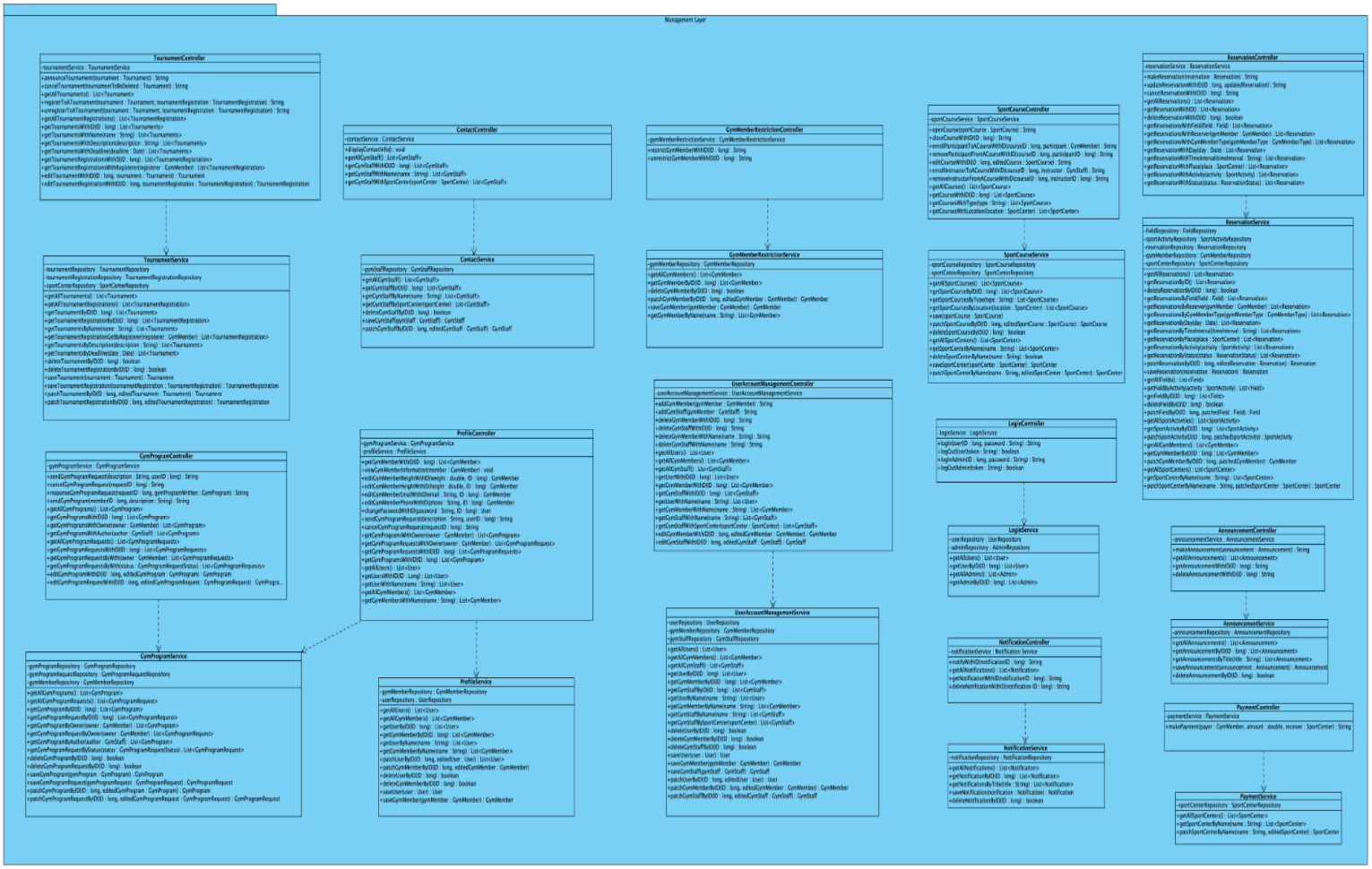

Fig. 5: Diagram of Classes in Management Layer (Click here for bigger size)

3.2.3 Database Layer

3.2.3.1 Database

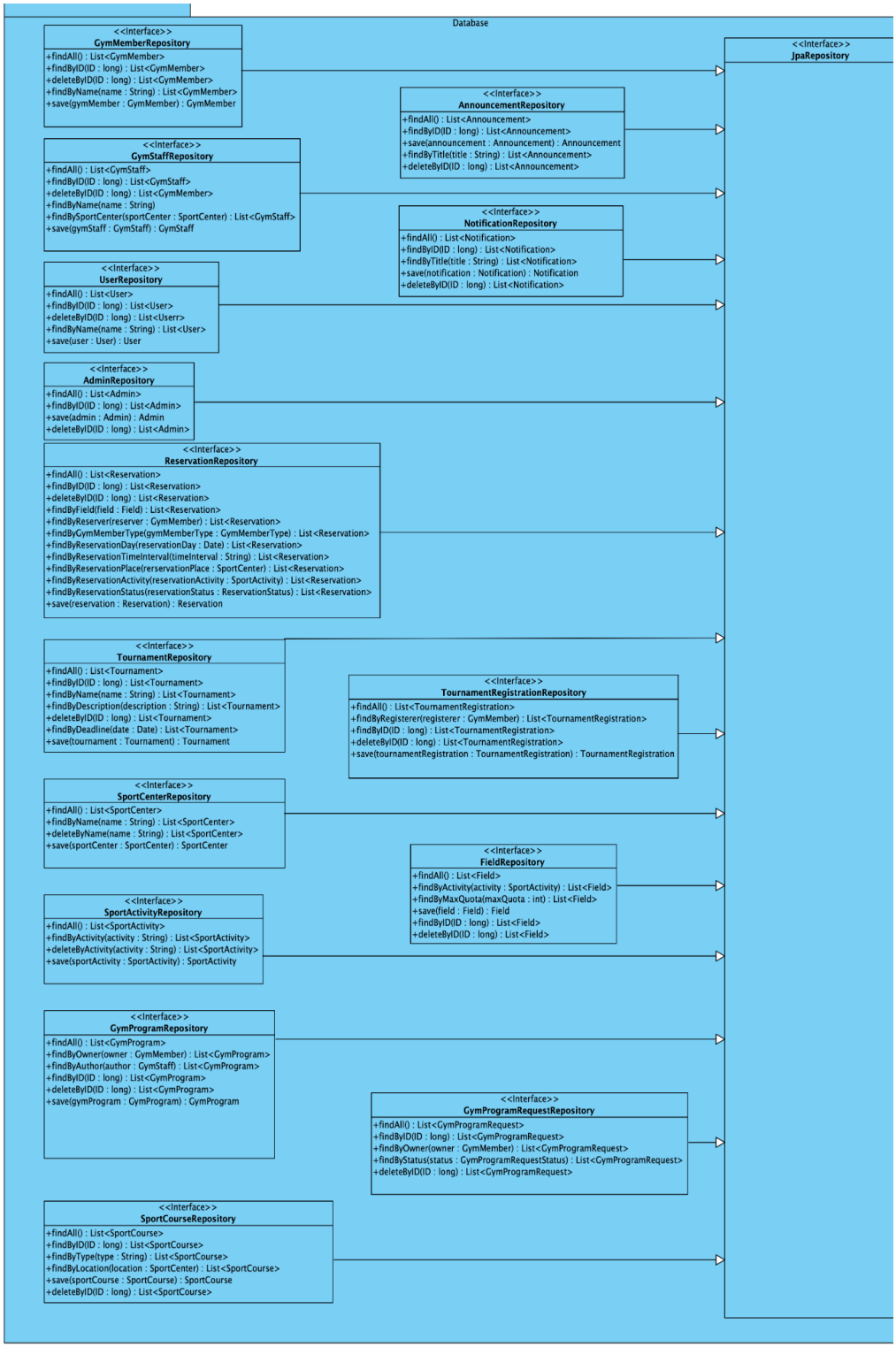

Fig. 6: Diagram of Classes of Database Package in Database Layer (Click here for bigger size)

3.2.3.2 Entity

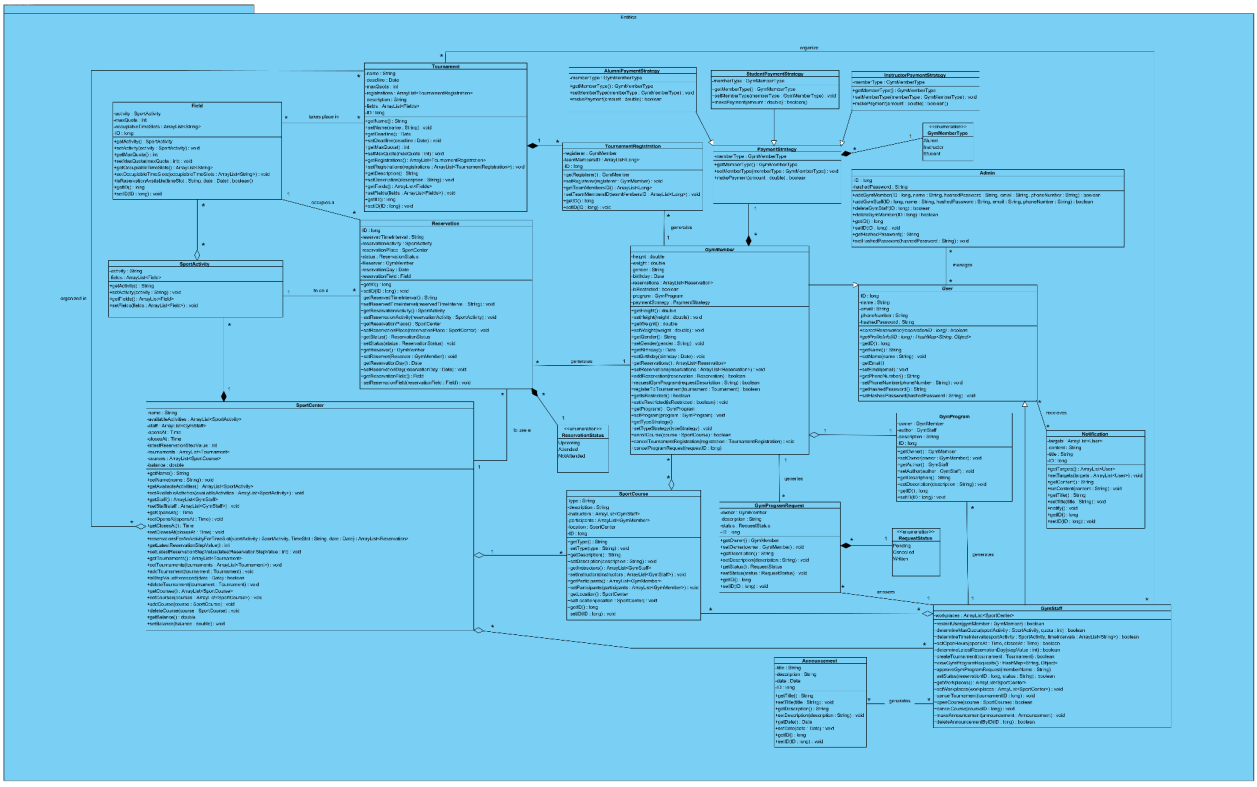

Fig. 7: Diagram of Classes of Entity Package in Database Layer (Click here for bigger size)

3.2.3.3 Design Patterns

We use Model-View-Controller (MVC) design pattern throughout the project by implementing our Model and Controller classes in Spring Boot application as well as implementing View classes in React web application. Their separation of concerns utilized our development process a lot since we are not bound to views to test our backend logic. Also, this kind of pattern usage helped the maintenance of the app with respect to specific concerns, backend or frontend.

Also, in our models, whenever it is necessary, we use Strategy and Singleton design patterns. For example, There are 3 types of Gym Members such as Student, Instructor, and Alumni. Some of those types have to pay differently with respect to some variables such as time and place. Their payment strategies build our Strategy pattern. This kind of pattern choice made our code clean because of the absence of nested if blocks. Also, we used a singleton design pattern in the instances of strategy objects as well as some helper classes. This design pattern prevents our

application from having different instances of doing the same business. It allows our memory to be used efficiently.

3.2.4 Final Object Diagram

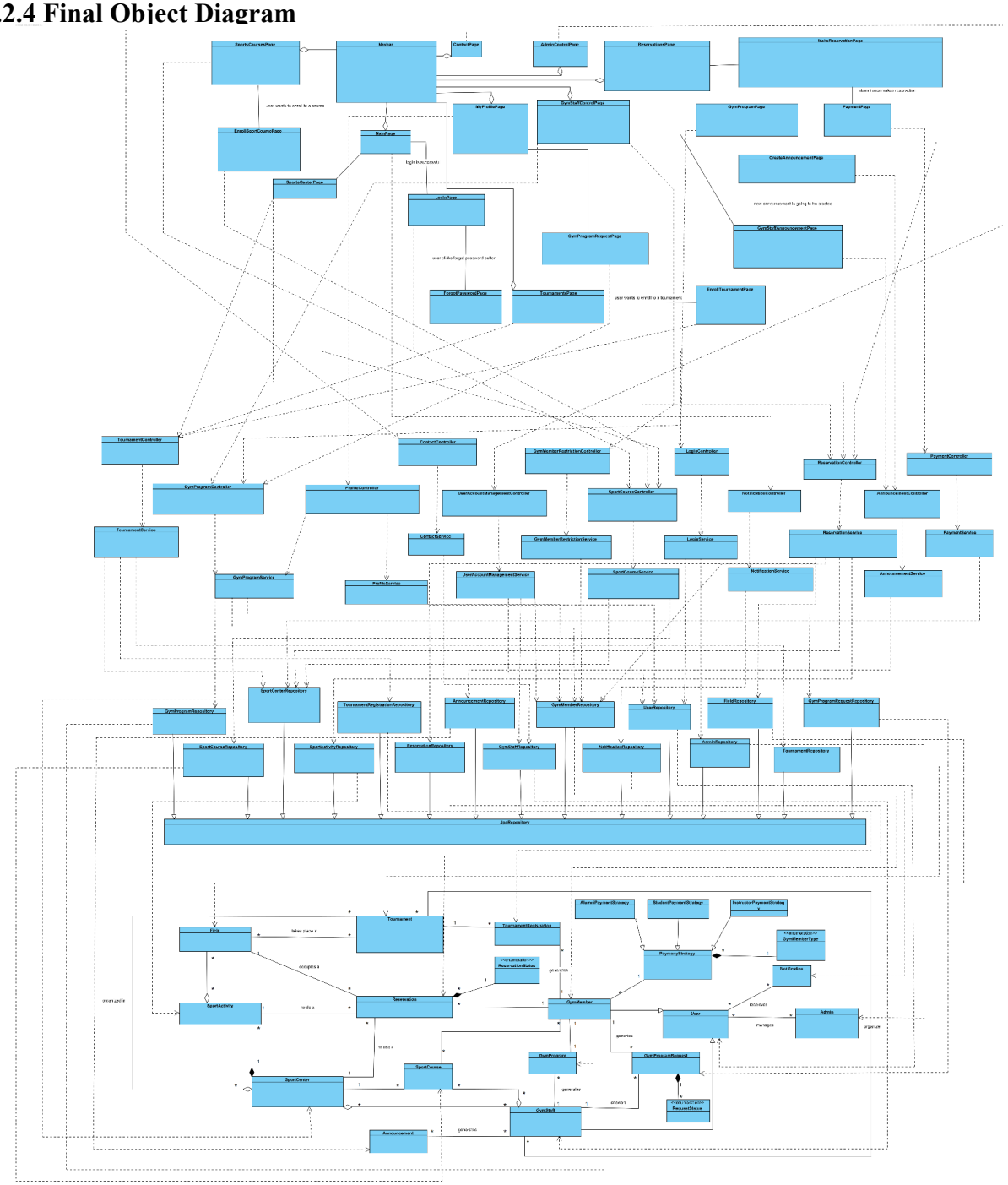

Fig. 8: Final Object Diagram (Click here for bigger size)

3.3 Packages

In our software, there are two kinds of packages. The first one is the internal packages which we develop, and the second one is the packages that are obtained by external APIs.

3.3.1 Internal Packages

These are the packages under the folder *com.<our_group_name>.bilfit*. In general, we have a hierarchy between the model, repository, service, controller, and view. That is like an enhanced version of Entity, Controller, and Boundary objects that we learned in class.

3.3.1.1 Model

This package contains the entities that are shown in our class diagram. They have instance variables, getters, setters, and some additional methods.

3.3.1.2 Repository

This package contains the repository interfaces for our models (entities). Thanks to Spring Boot and Java Persistence API, predefined findByX and deleteByX, where X is an instance of the model, operations on the models are automatically declared without a need for a concrete class. Therefore, they are interfaces.

3.3.1.3 Service

This package contains the services for major operations such as User Management, Gym, Tournament, GymProgram, Course, Profile, etc. in our application. For each operation, an interface and a class are needed. Interfaces define the needed method, and classes implement those interfaces by using repository instance variables.

3.3.1.4 Controller

This package contains the controller objects for the services. The controller generates an HTTP request endpoint for each service operation. The endpoints will be used to connect the backend and the frontend of the application.

3.3.1.5 View

This package contains the view objects for our application. The view objects are JavaScript files that are written in the React.js framework. The controllers' endpoints will be used to fetch data from our backend.

3.3.2 External Packages

External packages are the packages that we utilize during our project's development.

3.3.2.1 Spring Boot MySQL Driver

This API allows us to communicate with our MySQL database from Object-Oriented Java code without writing complex SQL queries.

3.3.2.2 Spring Web

This API allows us to create a restful web application with a spring boot project.

3.3.2.3 Spring Data JPA (Java Persistence API)

This API introduces some annotations to create endpoints for our controllers. This is as well working with MySQL database.

3.3.2.4 Spring Boot OAuth2 Client

This API makes our users log in, log out, and reset their password in our web application.

3.3.2.5 MaterialUI API

This API introduces a bunch of pre-designed UI elements in React.js framework. For a faster and more reliable development process.

3.4 Class Interfaces

3.4.1 Interface Layer Explanation

All of the classes are implemented using JavaScript. Types of parameters are written to specify their intended types.

- **NavBar Class**

This class is seen on every page, and it directs the user to the basic pages of the application. Also, users can log out with the logout button in the navbar.

Operations:

function goToReservation(): This function directs the user to the reservation page.

function goToMyProfile(): This function directs the user to my profile page.

function goToTournaments(): This function directs the user to the tournaments page.

function goToContact(): This function directs the user to the contact page.

function goToMainPage(): This function directs the user to the main page.

function logOut(): This function enables user to log out.

function goToGymStaffControlPage(isAuthorized: boolean): This function directs the user to the control page and it sends boolean whether user is authorized or not.

function goToAdminControlPage(isAuthorized: boolean): This function directs the user to the admin control page and it sends boolean whether user is authorized or not.

function goToSportsCoursePage(): This function directs the user to the sport courses page.

- **MainPage Class**

This class is the home of our web application. When users successfully log in, they are directed to this page.

Operations:

function viewAnnouncements(): This function enables users to view announcements added by gym staff.

function goToSportCenterPage(): This function enables users to go to the sports center page if they want to see the details of sports centers.

- **SportCenterPage Class**

This class shows the sports centers' details.

Operations:

function viewSportsCenterInformation(): This function enables users to view sport centers' facilities and rules.

- **LogInPage Class**

When users open the application, firstly, the LogInPage class appears because users must have accounts in order to benefit from the features of the application.

Operations:

function getID(): This function takes id input from the user.

function getEmail(): This function takes email input from the user.

function getPassword(): This function takes password input from the user.

function submitLoginInfoID(id: long, password: string): This function submits the inputs with the id coming from the user and it sends them to LogInManagement.

function submitLoginInfoEmail(email: string, password: string): This function submits the inputs with the id coming from the user and it sends them to LogInManagement.

function forgotPassword(): This function sends users to forgotPasswordPage class if users forget their passwords and click the button.

- **ForgotPasswordPage Class**

This class will be responsible for displaying the necessary items needed to reset the password of the user.

function getEmail(): This function will get the email input specified by the user.

function submitEmail(): This function will get the email specified in the above function and send it to the login controller class.

function goToLoginPage(): This function will redirect the user to the login page after the user requests a change of password.

- **AdminControlPage Class**

This class will be responsible for displaying necessary items that are needed to create and delete a user (for Admin).

Operations:

function viewUserList(): This function will provide the list of every user that is registered to the system.

function viewUserAdditionSection(): This function will display the necessary items (such as input blocks) that are needed to create a user.

function viewUserDeletionSection(): This function will display the necessary buttons that enable the Admin to delete the desired user.

- **ContactPage Class**

This class provides users to contact gym staff via email or phone numbers.

Operations:

function viewContactEmail(): This function provides users to view authorized staff's emails.

function viewContactPhoneNumber(): This function provides users to view authorized staff's phone numbers.

function viewAuthorizedStaffList(): This function provides users to view authorized staff's names.

- **MyProfilePage Class**

This class provides users to view and edit their personal information. Also, users can access program requests via the "request gym program" button.

Operations:

function viewWeight(): This function provides users to view their weights.

function viewHeight(): This function provides users to view their heights.

function viewPhoneNumber(): This function provides users to view their phone numbers.

function viewEmail(): This function provides users to view their emails.

function editProfile(): This function provides users to edit their profile information.

function goToChangePassword(): This function provides users to change their passwords via "change my password" button.

function goToRequestGymProgramPage(): This function directs users to subpage to request gym program.

- **GymProgramRequestPage Class**

This class enables users to request or cancel programs that are written by gym staff.

Operations:

function getGymProgramDescription(): This function enables users to write description about their desired program.

function submitGymProgramDescription(description: string, userId: long): This function takes description and userID as parameters. Then, it sends them to GymProgramManagement class.

function viewRequestedProgramList(): This function enables users to view their past request list.

function cancelRequestedProgram(gymProgramID: long): This function enables users to cancel their waiting request.

- **GymStaffControlPage Class**

This class can be seen only by gym staff to view all gym members, edit their restriction status and make announcements.

Operations:

function viewRestrictedUsers(): This function enables gym staff to view all restricted gym members.

function viewUserList(): This function enables gym staff to view all gym members.

function getSearchedUser(userID: long): This function enables gym staff to search specific gym member.

function goToGymProgramPage(): This function directs gym staff to GymProgramPage.

function editRestriction(id: long, isRestricted: boolean): This function enables gym staff to restrict the gym member in order not to make reservations.

function goToGymStaffAnnouncementPage(): This function directs gym staff to GymStaffAnnouncementPage.

- **GymStaffAnnouncementPage Class**

This class will display content for the gym staff that is necessary for the management of an announcement.

Operations:

function viewAllAnnouncements(): The staff will be able to view every announcement that is created.

function deleteAnnouncement(announcement: Announcement): Staff will be able to remove any announcement that they see fit. This method will check the click of the button and then invoke the announcement controller.

function editAnnouncement(announcement: Announcement): Staff will also be able to make edits on the announcements content, date, and any other specification.

function goToCreateAnnouncementPage(): If the staff wants to create a new announcement, they will be redirected to a new page with this function.

- **CreateAnnouncementPage Class**

This class will be responsible for displaying the necessary tools that are needed to create an announcement.

function getTitle(): This function will get input of the title of the announcement from staff.

function getDescription(): This function will get input of the description of the announcement from staff.

function submitNewAnnouncement(title: string, description: string): This function will get the previously specified inputs and will send them to the announcement controller.

- **GymProgramPage Class**

This class enables gym staff to view and manage gym programs.

Operations:

function viewGymProgramRequests(): This function enables gym staff to view all gym program requests.

function getGymDescription(): This function enables gym staff to write their programs to users.

function submitGymProgram(userID: long, program: string): This function submit gym staff's program and sends to GymProgramController.

function cancelGymProgramRequest(): This function enables gym members to cancel their gym program requests.

- **TournamentsPage Class**

This class enables users to view active tournaments, and gym staff can manage tournaments on this page.

Operations:

function viewActiveTournaments(): This function enables users to view all active tournaments.

function goToEnrollTournamentPage(): This function directs gym members to goToEnrollTournamentPage.

function cancelEnrolledTournaments(tournamentID: long): This function enables users to cancel their enrolled tournaments. In order to cancel them. This function sends tournamentID TournamentController class.

function viewAllTournaments(isAuthorized: boolean): This function enables gym staff to view all active tournaments.

function addNewTournament(isAuthorized: boolean): This function enables gym staff to add new tournament.

function cancelTournament(isAuthorized: boolean): This function enables gym staff to cancel active tournaments.

function viewAllUserTournamentRegistrations(isAuthorized: boolean): This function enables gym staff to view all active tournaments registrations.

function editUserTournamentRegistrations(isAuthorized: boolean): This function enables users to edit registrations.

- **EnrollTournamentPage Class**

This class enables gym members to enroll in an available tournament.

Operations:

function getTournamentType(): This function enables gym member to choose tournament type.

function getTournamentLocation(): This function enables gym member to choose tournament location.

function getMemberInfo(): This function enables gym members to add themselves and their friends to tournaments.

function getTime(): This function enables gym member to choose tournament time.

function getDate(): This function enables gym member to choose tournament date.

function submitEnrollmentInfo(tournamentType: Tournament, location: Field, time: Time, date: Date, member: ArrayList<GymMember>): This function submits the inputs and calls TournamentController class.

- **SportsCoursesPage Class**

This class enables users to view available sports courses and directs them if they want to enroll them.

Operations:

function viewActiveCourseDetails(): This function enables users to view all active courses and their details.

function goToEnrollSportCoursePage(): This function directs users to EnrollSportCoursePage if they want to enroll a sport course.

function cancelEnrolledSportCourse(courseID: long): This function enables users to cancel their enrollments by calling SportCourseController.

function viewAllUserCourses(isAuthorized: boolean): This function enables gym staff to view all active courses and their enrollments.

function addNewCourse(isAuthorized: boolean): This function enables gym staff to add new sport course by calling SportCourseController.

function cancelCourse(isAuthorized: boolean): This function enables gym staff to remove sport course by calling SportCourseController.

- **EnrollSportCoursePage Class**

This class enables gym members to enroll in an available sports course.

Operations:

function getSportCourseType(): This function enables users to choose sport course type.

function getCourseLocation(): This function enables users to choose sport course location.

function getTime(): This function enables users to choose sport course time.

function getDate(): This function enables users to choose sport course date.

function submitCourseEnrollmentInfo(courseType: string, location: SportCenter, time: Time, date: Date, user: GymMember): This function submit inputs and call SportCourseController.

- **ReservationsPage Class**

The class is responsible for displaying and getting input about reservation-related content such as viewing reservation list and managing those reservations.

Operations:

function viewReservations(): This function will be responsible for displaying the reservations list and also it will have a button to go to the "Make Reservations" subpage.

function cancelReservations(): Responsible for canceling the current reservations that the user has already made, in case the user will not be able to come or other factors.

function viewAllUserReservations(isAuthorized: boolean): This function is specifically made for the gym staff. Since the gym staff is responsible for managing the reservations of the gym members, they will have functionality that enables them to see all of the reservations made by the members.

function goToMakeReservationPage(): In order to make a reservation, the user will press a button and they will be redirected to a make reservations subpage.

function editAllUserReservations(isAuthorized: boolean): This is a gym staff specific method. Gym staff will be able to edit the reservations of all users. They have the ability to cancel and change the date of the reservations if needed.

function deletePastReservationsFromList(reservationID: long): The user will be able to see every single reservation that they have made. They may want to delete the past reservations in order to free up space, which is the reason this function is existing.

- **MakeReservationPage Class**

It will be responsible for displaying the content of displaying and getting input related to information about the desired reservation.

Operations:

function getSportCenter(): It will get input about which Sports Center the user wants to make a reservation (East Campus, Main Campus…)

function getSportActivityPlace(): The user will be able to have reservations in different places (e.g. different courts for basketball) and this class will get the input by using this method.

function getTimeSlot(): It will get the specified time slot of the reservation.

function getDate(): It will get the specified date of the reservation (At max 2 days later than the current date)

function getSportActivity(): This function will get the input about the type of the sport activity.

function goToPaymentMethod(userType: GymMemberType): This function is specific for alumni users. They need to pay before making a reservation and therefore, this function will direct them to the payment page.

function submitReservationInfo(sportCenter: SportCenter, timeSlot: Time, date:Date, sportActivity: SportActivity): This method will be responsible for getting every input specified in the previous methods and sending all the information as a whole to the reservation management package in the management layer.

- **PaymentPage Class**

Operations:

This class will be responsible for displaying and getting input about payment related data.

function getCreditCardInfo(): It will get the credit card info that is specified by the user.

function submitPayment(creditInfo: string): The user will click a button, and then this function will get the specified payment info and send it to the payment controller class.

3.4.2 Management Layer Explanation

In the management layer, we have two objects for each major operation in our system such as Reservation, User Management, Gym, Tournament, GymProgram, Course, Profile, etc. These two objects are *Controller* and *Service* Objects. Service objects access the needed data from Repository objects that are discussed on the database layer. Controller objects use Services to create endpoints for outside access to the system. For example, GET, POST, and PATCH requests are created in those objects for each major operation.

In general, we almost always have the following methods in service objects:

getAll() : List<Object> : Returns the list of all objects that are instances of the class Object.

getByID(ID : long) : List<Object> : Returns the object whose ID is specified as list. However, ID is unique among all classes. Therefore, at least 0 at most 1 object is returned. We decided to return as a list since dealing with empty lists is easier than dealing with null objects.

deleteByID(ID : long) : boolean : This function returns true if the ID specified is on the repository after deleting the object with that ID.
patchByID(ID : long, newObj : Object) : Object : This function edits the object on the repository whose id is given.
save(obj : Object) : Object : This function creates a new instance of an object in the repository.

In controller objects, we defined methods that can make necessary changes in the database and user interface to perform functional requirements. Controller objects depend on service objects to do these wanted changes. Some examples of the methods we defined in control objects are:
makeReservation(reservation: Reservation) : String : Adds a reservation for activity and field in the system. It takes a Reservation object as a parameter, and it performs necessary changes in ReservationRepository and FieldRepository by using ReservationServices with ReservationController. It returns a String which informs about the success of the process.
enrollParticipantToACourse(courseID: long, participant: GymMember) : String : Adds a new participant for a sport course in the system. It takes the new participant as a GymMember object and takes the ID of the target course. It performs necessary changes in the SportCourseRepository by using SportCourseServices with SportCourseController. It returns a String which informs about the success of the process.
restrictGymMemberWithID(ID: long) : String : When a GymStaff wants to restrict a GymMember to prevent it from making reservations, this method in the GymMemberRestrictionController is called with the ID of that GymMember. It makes the necessary changes in GymMemberRepository by using GymMemberRestrictionService. It returns a String which informs about the success of the process.

3.4.3 Database Layer Explanation

All of the classes are implemented using Java and the Spring Boot framework.

- **GymMemberRepository Interface**

This class is a Spring repository used to take care of various database-related behaviours and functions that can be useful for a Gym Member object.

function findAll(): This function returns a list of all the Gym Member users available in the database. If there is no match, it will return an empty list.
function findByID(ID : long): This function returns a list of all the Gym Member users with the given ID attribute as an input. If there is no match, it will return an empty list.
function deleteByID(ID : long): This function returns a list of all the Gym Member users with the given ID attribute as input and deletes the matching ones from the database.
function findByName(name : String): This function returns a list of all the Gym Member users with the given name attribute as an input. If there is no match, it will return an empty list.
function save(gymMember : GymMember): This function saves a new Gym Member user to the database.

- **AnnouncementRepository Interface**

This class is a Spring repository used to implement various operations related to the Announcement objects' storage in the database.
function findAll(): This function returns a list of all the Announcement objects available in the database. If there is no match, it will return an empty list.
function findByID(ID : long): This function returns a list of all the Announcement objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function save(announcement : Announcement): This function saves a new Announcement object to the database.
function findByTitle(title : String): This function returns a list of all the Announcement objects with the given title attribute as an input. If there is no match, it will return an empty list.
function deleteByID(ID : long): This function returns a list of all the Announcement objects with the given ID attribute as input and deletes the matching ones from the database.

- **GymStaffRepository Interface**

This class is a Spring repository used to implement various behaviours that can be useful for a Gym Staff object in relation with the database.
function findAll(): This function returns a list of all the Gym Staff users available in the database. If there is no match, it will return an empty list.

function findByID(ID : long): This function returns a list of all the Gym Staff users with the given ID attribute as an input. If there is no match, it will return an empty list.
function deleteByID(ID : long): This function returns a list of all the Gym Staff users with the given ID attribute as input and deletes the matching ones from the database.
function findByName(name : String): This function returns a list of all the Gym Staff users with the given name attribute as an input. If there is no match, it will return an empty list.
function findBySportCenter(sportCenter : SportCenter): This function returns a list of Gym Staff users that are working on a specific Sports Center which is given as an input. If there is no match, it will return an empty list.
function save(gymStaff : GymStaff): This function saves a new Gym Member user to the database.

- **NotificationRepository Interface**

This class is a Spring repository used to implement various operations related to the Notification objects' storage in the database.
function findAll(): This function returns a list of all the Notification objects available in the database. If there is no match, it will return an empty list.
function findByID(ID : long): This function returns a list of all the Notification objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function findByTitle(title : String): This function returns a list of all the Notification objects with the given title attribute as an input. If there is no match, it will return an empty list.
function save(notification : Notification): This function saves a new Notification object to the database.
function deleteByID(ID : long): This function returns a list of all the Notification objects with the given ID attribute as input and deletes the matching ones from the database.

- **UserRepository Interface**

This class is a Spring repository used to implement various behaviors that can be useful for a User object in relation to the database.
function findAll(): This function returns a list of all the User objects available in the database. If there is no match, it will return an empty list.

function findByID(ID : long): This function returns a list of all the User objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function deleteByID(ID : long): This function returns a list of all the User objects with the given ID attribute as an input and delete the matching ones from the database.
function findByName(name : String): This function returns a list of all the User objects with the given name attribute as an input. If there is no match, it will return an empty list.
function save(user : User): This function saves a new User object to the database.

- **AdminRepository Interface**

This class is a Spring repository used to implement various behaviors and functions that can be useful for an Admin object in relation to the database.
function findAll(): This function returns a list of all the Admin objects available in the database. If there is no match, it will return an empty list.
function findByID(ID : long): This function returns a list of all the Admin objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function save(admin : Admin): This function saves a new Admin object to the database.
function deleteByID(ID : long): This function returns a list of all the Admin objects with the given ID attribute as input and deletes the matching ones from the database.

- **ReservationRepository Interface**

This class is a Spring repository used to take care of various database-related behaviors and functions that are to be used frequently.
function findAll(): This function returns a list of all the Reservation objects available in the database. If there is no match, it will return an empty list.
function findByID(ID : long): This function returns a list of all the Reservation objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function deleteByID(ID : long): This function returns a list of all the Reservation objects with the given ID attribute as input and deletes the matching ones from the database.
function findByField(field : Field): This function returns a list of all the Reservation objects whose field is the specified input. If there is no match, it will return an empty list.

function findByReserver(reserver : GymMember): This function returns a list of all the Reservation objects with the given reserver as an input, who is also a Gym Member object. If there is no match, it will return an empty list.
function findByGymMemberType(gymMemberType : GymMemberType): This function returns a list of all the Reservation objects whose reserver is the given Gym Member Type. If there is no match, it will return an empty list.
function findByReservationDay(reservationDay : Date): This function returns a list of all the Reservation objects with the given reservation day which is a date input. If there is no match, it will return an empty list.
function findByReservationTimeInterval(timeInterval : String): This function returns a list of all the Reservation objects with the given time interval as an input. If there is no match, it will return an empty list.
function findByPlace(reservationPlace : SportCenter): This function returns a list of all the Reservation objects with a Sports Center input denoting for which sports center the reservation was done. If there is no match, it will return an empty list.
function findByReservationActivity(reservationActivity : SportActivity): This function returns a list of all the Reservation objects with the given sports activity the reservation is done for as an input. If there is no match, it will return an empty list.
function findByStatus(reservationStatus : ReservationStatus): This function returns a list of all the Reservation objects with the given status as an input. If there is no match, it will return an empty list.
function save(reservation : Reservation): This function saves a new Reservation object to the database.

- **TournamentRepository Interface**

This class is a Spring repository used to implement various behaviors that can be useful for a Tournament object in relation with the database.
function findAll(): This function returns a list of all the Tournament objects available in the database. If there is no match, it will return an empty list.
function findByID(ID : long): This function returns a list of all the Tournament objects with the given ID attribute as an input. If there is no match, it will return an empty list.

function findByName(name : String): This function returns a list of all the Tournament objects with the given name attribute as an input. If there is no match, it will return an empty list.
function findByDescription(description: String): This function returns a list of all the Tournament objects whose descriptions match the given input. If there is no match, it will return an empty list.
function deleteByID(ID : long): This function returns a list of all the Tournament objects with the given ID attribute as input and deletes the matching ones from the database.
function findByDeadline(date: Date): This function returns a list of all the Tournament objects whose deadline is the specified date. If there is no match, it will return an empty list.
function save(tournament : Tournament): This function saves a new Tournament object to the database.

- **TournamentRegistration Interface**

This class is a Spring repository used to implement various behaviors that can be useful for a TournamentRegistration object in relation to the database.
function findAll(): This function returns a list of all the TournamentRegistration objects available in the database. If there is no match, it will return an empty list.
function findByRegisterer(registerer: GymMember): This function returns a list of all the TournamentRegistration objects whose registerer is the given Gym Member user. If there is no match, it will return an empty list.
function findByID(ID : long): This function returns a list of all the TournamentRegistration objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function deleteByID(ID : long): This function returns a list of all the TournamentRegistration objects with the given ID attribute as input and deletes the matching ones from the database.
function save(tournamentRegistration : TournamentRegistration): This function saves a new TournamentRegistration object to the database.

- **SportCenterRepository Interface**

This class is a Spring repository used to implement various operations related to the Sport Center objects' storage in the database.

function findAll(): This function returns a list of all the Sport Center objects available in the database. If there is no match, it will return an empty list.
function findByName(name : String): This function returns a list of all the Sports Center objects with the given name attribute as an input. If there is no match, it will return an empty list.
function deleteByName(name: String): This function returns a list of all the Sports Center objects with the given ID attribute as input and deletes the matching ones from the database.
function save(sportCenter : SportCenter): This function saves a new Gym Member user to the database.

- **FieldRepository Interface**

This class is a Spring repository used to implement various database-related behaviors that can be useful for a Field object.
function findAll(): This function returns a list of all the Field objects available in the database. If there is no match, it will return an empty list.
function findByActivity(activity : SportActivity): This function returns a list of all the Field objects which are used for the specified input which is a Sport Activity object. If there is no match, it will return an empty list.
function findByMaxQuota(maxQuota : int): This function returns a list of all the Field objects that have the specified max quota. If there is no match, it will return an empty list.
function save(field : Field): This function saves a new Field object user to the database.
function findByID(ID : long): This function returns a list of all the Field objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function deleteByID(ID: long): This function returns a list of all the Field objects with the given ID attribute as input and deletes the matching ones from the database.

- **SportActivityRepository Interface**

This class is a Spring repository used to implement various behaviors that can be useful for a Sports Activity object in relation to the database.
function findAll(): This function returns a list of all the Sport Activity objects available in the database. If there is no match, it will return an empty list.

function findByActivity(activity : String): This function returns a list of all the Sport Activity objects with the given activity name. If there is no match, it will return an empty list.
function deleteByActivity(activity : String): This function returns a list of all the Sport Activity objects with the given activity name attribute as input and deletes the matching ones from the database.
function save(sportActivity : SportActivity): This function saves a new Sports Activity object user to the database.

- **GymProgramRepository Interface**

This class is a Spring repository used to implement various operations related to the Gym Program Request objects' storage in the database.
function findAll(): This function returns a list of all the Gym Program objects available in the database. If there is no match, it will return an empty list.
function findByOwner(owner : GymMember): This function returns a list of all the Gym Program objects whose owner is the specified Gym Member user. If there is no match, it will return an empty list.
function findByAuthor(author : GymStaff): This function returns a list of all the Gym Program objects whose author is the specified Gym Staff user. If there is no match, it will return an empty list.
function findByID(ID : long): This function returns a list of all the Gym Program objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function deleteByID(ID : long): This function returns a list of all the Gym Program objects with the given ID attribute as input and deletes the matching ones from the database.
function save(gymProgram : GymProgram): This function saves a new Gym Program object user to the database.

- **GymProgramRequestRepository Interface**

This class is a Spring repository used to implement various behaviors that can be useful for a Gym Program Request Repository object in relation to the database.
function findAll(): This function returns a list of all the Gym Program Request objects available in the database. If there is no match, it will return an empty list.

function findByID(ID : long): This function returns a list of all the Gym Program objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function findByOwner(owner : GymMember): This function returns a list of all the Gym Program objects whose owner is the specified Gym Member user. If there is no match, it will return an empty list.
function findByStatus(status : GymProgramRequestStatus): This function returns a list of all the Gym Program Request objects with the given status as an input. If there is no match, it will return an empty list.
function deleteByID(ID : long): This function returns a list of all the Gym Program Request objects with the given ID attribute as input and deletes the matching ones from the database.

- **SportCourseRepository Interface**

This class is a Spring repository used to implement various behaviors that can be useful for a Sports Course object in relation to the database.
function findAll(): This function returns a list of all the Sport Course objects available in the database. If there is no match, it will return an empty list.
function findByID(ID : long): This function returns a list of all the Sport Course objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function findByType(type : String): This function returns a list of all the Sport Course objects with the given type name. If there is no match, it will return an empty list.
function findByLocation(location : SportCenter): This function returns a list of all the Sport Course objects whose location is the specified Sports Center object. If there is no match, it will return an empty list.
function save(sportCourse : SportCourse): This function saves a new Sport Course object user to the database.
function deleteByID(ID : long): This function returns a list of all the Sport Course objects with the given ID attribute as input and deletes the matching ones from the database.

- **JpaRepository Interface**

JPARepository is the interface inherited by our repository interfaces to work with Java Spring Framework. It allows to easily generate SQL trees for entities in the database.

4. Improvement Summary

In the second iteration, we have made some additions according to the feedback gathered from our TA, Instructor, Sibling Group, and our own observations.

4.1 Design Goals

- More technical details are added to the Usability section in order to make it more appropriate for a design report.

4.2 High-Level Software Architecture

- Subsystem Decomposition explanations are added.
- Deployment Diagram is added.
- A textual description is added for all the 3 layers of the 3 Layer Architectural design.
- Hardware/Software Mapping is updated accordingly to represent the hardware details of the local machine that is going to be used to run the application.
- Persistent Data Management section is updated to give a better idea of all the data that are going to be stored in the database.
- Access Matrix is added.
- Boundary Conditions now is written by the viewpoint of the admin.

4.3 Low-Level Design

- Changed one tradeoff from the first iteration to Maintainability-Efficiency as it makes more sense for our application.
- New associations are added and some existing ones are updated in the Entity diagram.
- Textual descriptions of the design patterns used in the implementation of this application are added.
- Final Object Diagram is added.

5. Glossary & References

[1] N. Pandit, “What and why react.js,” *C# Corner*. [Online]. Available: https://www.c-sharpcorner.com/article/what-and-why-reactjs/. [Accessed: 03-Apr-2022].

[2] A. A. K. W. at B. Agile, Author: and W. at B. Agile, “Pros and cons of using Spring Boot,” *Insights*, 02-Oct-2021. [Online]. Available: https://bambooagile.eu/insights/pros-and-cons-of-using-spring-boot/. [Accessed: 04-Apr-2022].

[3] Oracle, “Introduction to Java Persistence API,” Oracle. [Online]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html. [Accessed: April 2, 2022].